Assignment 2

Problem Solving and Algorithm Design: Sudoku Solver Program

Pedram Pasandide

Due Date: 10 March

1 Introduction

In this assignment, you are tasked with writing a program to solve Sudoku puzzles. Sudoku is a logic-based combinatorial number-placement puzzle. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids that compose the grid contain all of the digits from 1 to 9. Solving a Sudoku puzzle in programming can be done using techniques like backtracking and recursion functions.

2 Programming Sudoku Auto Solver(8 points)

Your program should take an unsolved Sudoku puzzle as input and produce a solved Sudoku puzzle as output. The input is hard coded. Take a look at the following code. This is the general format that your code must have. The input `grid`, the Sudoku puzzle, is defined in the `int main()`.

```
// Code: Here include your necessary library(s)
// Code: Write your global variables here, like:
#define N 9

/*Code: write your functions here, or the declaration of the function/
For example write the recursive function solveSudoku(), like:*/
<Returned Value> solveSudoku(<input arguments>)
{
 // Code: count+1, the number of times the function was called.
 // Code: here write the implementation of solveSudoku
}

int main()
{
 // This is hard coding to receive the "grid"
 int grid[N][N] = {
  {0, 2, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 6, 0, 0, 0, 0, 3},
  {0, 7, 4, 0, 8, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 3, 0, 0, 2},
```

```
    {0, 8, 0, 0, 4, 0, 0, 1, 0},
    {6, 0, 0, 5, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 7, 8, 0},
    {5, 0, 0, 0, 0, 9, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 4, 0}};

  // For more samples to check your program, google for solved samples, or
  // check https://sandiway.arizona.edu/sudoku/examples.html

  printf("The input Sudoku puzzle:\n");
  // "print" is a function we define to print the "grid"
  print(grid);

  if (solveSudoku(<input arguments>))
  {
   // If the puzzle is solved then:
   printf("Solution found after %d iterations:\n", count);
   print(grid);
  }
  else
  {
   printf("No solution exists.");
  }
  return 0;
}
/*Code: If you have functions that are declared but not implemented
they, here write the implementation.*/
```

Follow the comments, and write your code in a file named `Sudoku_Solver.c`. If you need more solved puzzle check this website. If for any reason your program was stuck in an infinite loop, go to the terminal running the program, press `Cnrtl+C` to force stopping the execution (kill the program). After compiling and running your code, this is what I have to see in the terminal:

```
The input Sudoku puzzle:
0 2 0 | 0 0 0 | 0 0 0 |
0 0 0 | 6 0 0 | 0 0 3 |
0 7 4 | 0 8 0 | 0 0 0 |
----------------------
0 0 0 | 0 0 3 | 0 0 2 |
0 8 0 | 0 4 0 | 0 1 0 |
6 0 0 | 5 0 0 | 0 0 0 |
----------------------
0 0 0 | 0 1 0 | 7 8 0 |
5 0 0 | 0 0 9 | 0 0 0 |
```

```
0 0 0 | 0 0 0 | 0 4 0 |
----------------------
Solution found after 112124 iterations:
1 2 6 | 4 3 7 | 9 5 8 |
8 9 5 | 6 2 1 | 4 7 3 |
3 7 4 | 9 8 5 | 1 2 6 |
----------------------
4 5 7 | 1 9 3 | 8 6 2 |
9 8 3 | 2 4 6 | 5 1 7 |
6 1 2 | 5 7 8 | 3 9 4 |
----------------------
2 6 9 | 3 1 4 | 7 8 5 |
5 4 8 | 7 6 9 | 2 3 1 |
7 3 1 | 8 5 2 | 6 4 9 |
----------------------
```

The first input puzzle is the initial input, and the second one shows the solved Sudoku. If the puzzle could not be solved it will print `No solution exists.`.

3 Graphical User Interface (+1 bonus)

To program this section, you have to go beyond the lecture notes. GUI, or Graphical User Interface, is a visual way for users to interact with software applications through graphical elements like windows, buttons, menus, and icons, rather than using text-based commands. GUIs make applications more user-friendly, intuitive, and visually appealing by providing a familiar environment for users to navigate.

GTK, which stands for GIMP Toolkit, is a widely used open-source toolkit for creating graphical user interfaces. It's written in C but provides bindings for many other programming languages, making it versatile. GTK offers a simple and efficient way to design and develop GUI applications in C language. It provides a wide range of widgets and tools for building interactive interfaces, making it suitable for both small and large-scale projects.

Read the `COMPSCI1XC3_GTK.pdf` and learn how to work with `Gtk_example.c`, which is a simple example of using GTK in GUIs. Then try to implement the algorithm you developed from the previous section, within a graphical interface. Write all your codes in `Sudoku_Solver_GUI.c`. I have to be able to compile your code with:

```
gcc -o Soduku Sudoku_Solver_GUI.c `pkg-config --cflags --libs gtk+-3.0
```

And run the application `Soduku` using `./Soduku` in the terminal to open the application. For example, if I want to solve the following puzzle:

```
0 2 0 | 0 0 0 | 0 0 0 |
0 0 0 | 6 0 0 | 0 0 3 |
0 7 4 | 0 8 0 | 0 0 0 |
----------------------
0 0 0 | 0 0 3 | 0 0 2 |
0 8 0 | 0 4 0 | 0 1 0 |
6 0 0 | 5 0 0 | 0 0 0 |
----------------------
0 0 0 | 0 1 0 | 7 8 0 |
5 0 0 | 0 0 9 | 0 0 0 |
0 0 0 | 0 0 0 | 0 4 0 |
----------------------
```

Then using a graphical interface, I can give the inputs to the application, like:

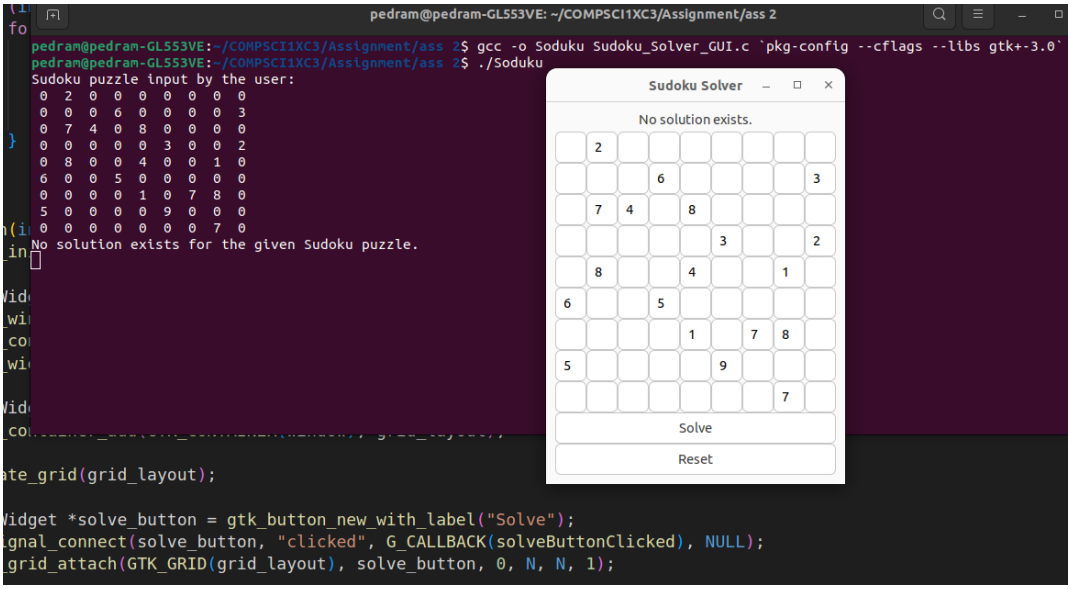

Figure 1: The application created with the name **Sudoku Solver**.

Here, I clicked on "**Solve**" button, and at the top of the application it says: "**No solution exists.**". The solver couldn't solve the problem, because in the last row, the second column from right, the number should be `4` not `7`. At the same time, the application is printing some stuff in the terminal that software was run. But at this point it doesn't matter anymore, because the user is not supposed to see what is going on behind the software. This print might be useful during debugging and developing the program.

After removing 7 and entering number 4, then I press **Solve**, and I can see the solved puzzle. The numbers in red color are those we gave to the application:

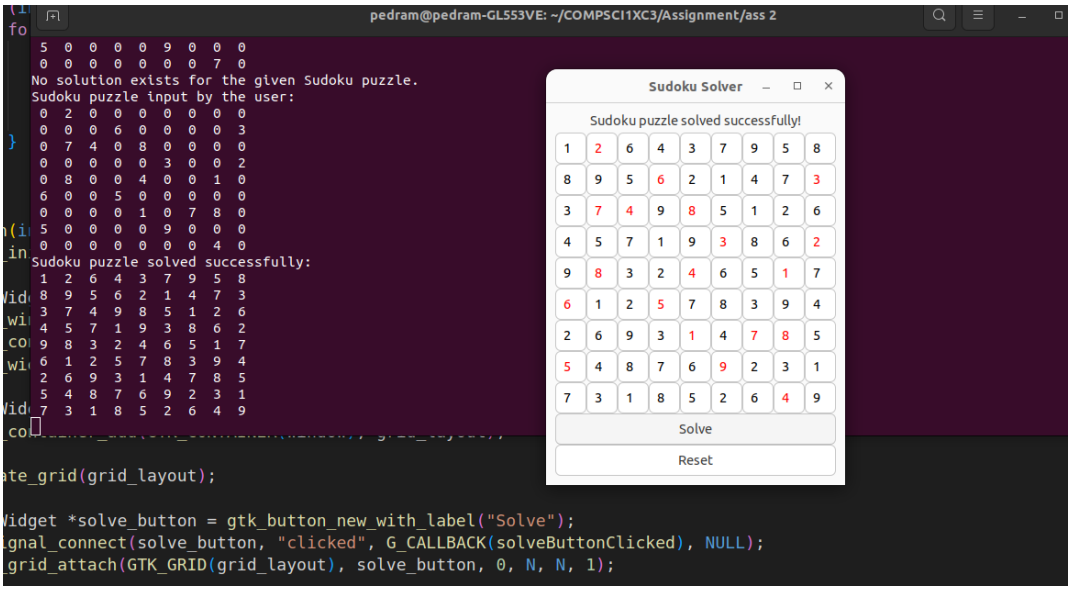

Figure 2: After fixing the wrong entry and clicking on **Solve** button.

It makes sense to create a button to reset the game instead of closing all the game, and opening it again, or even removing the values one by one. So I created **Reset** button:

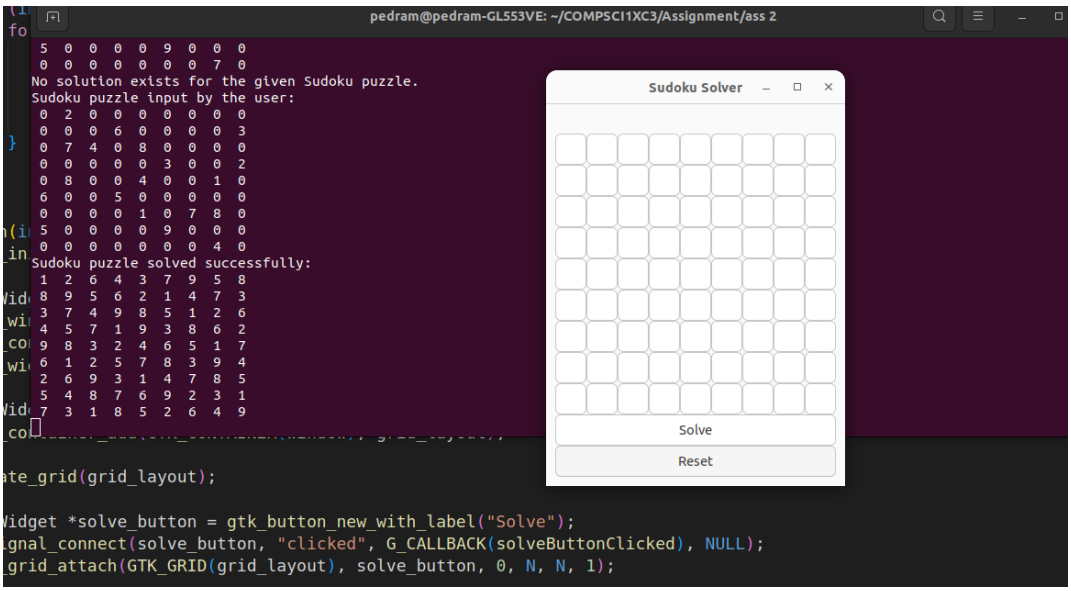

Figure 3: After clicking on **Reset** button.

I can keep playing like:

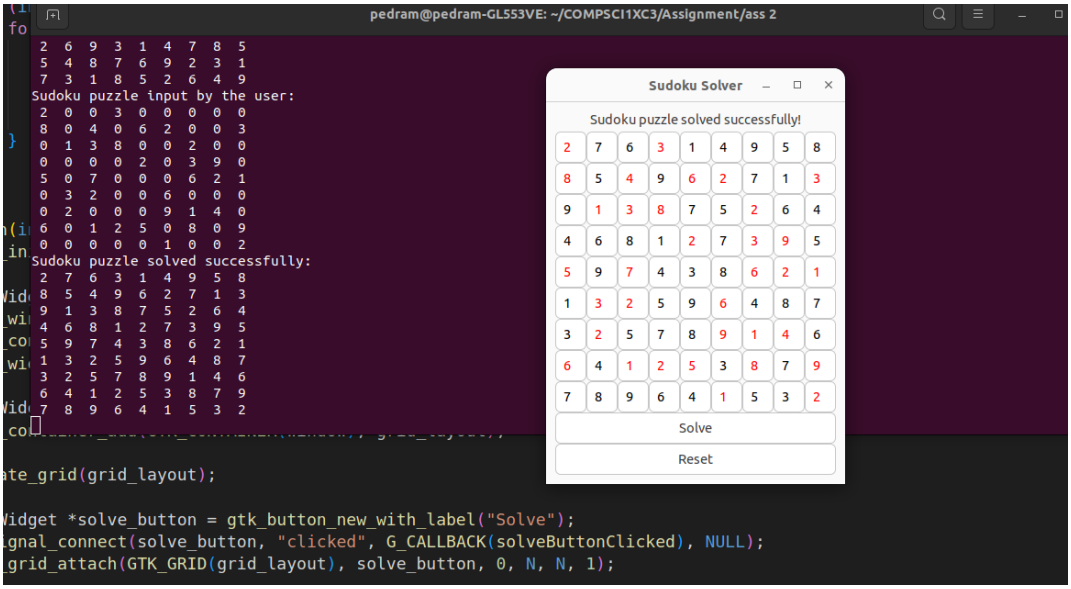

Figure 4: Another puzzle after **Reset**.

My application will stop working when I close the window opened. When you close the window, make sure the program has stopped from the terminal too. Press `Cnrtl+C` in the terminal to force stopping the execution (kill the program) if necessary. You can add a little bit more stuff to make it more graphically beautiful or self presenting. Maybe for fun in the future, you can receive the inputs from user by screenshots or images. If you are looking for even more fun, you can do it for many popular, simple, and puzzle games! However, don't forget, C is not the best language for GUIs.

4 Report (2 points)

Your report must be in a LaTeX format (`report.tex`). In your report, describe the algorithm you have programmed, the purpose of each function, and include all your codes in `Sudoku_Solver.c` **or** `Sudoku_Solver_GUI.c`. Produce the PDF file (`report.pdf`) to make sure it is working, and submit both of them.

5 Submission On Avenue to Learn

You can use any resources to write the code. Don't forget to mention the source, and to follow the submission guidelines. No zip files on Avenue. Please avoid copying. If the copied percentage exceeds the class average, you may be required to present your code.

- `Sudoku_Solver.c` **OR** `Sudoku_Solver_GUI.c`
- `report.tex` **AND** `report.pdf`